University of Utah
Electrical & Computer Engineering Department
ECE 3510 Lab 8
Ball and Beam

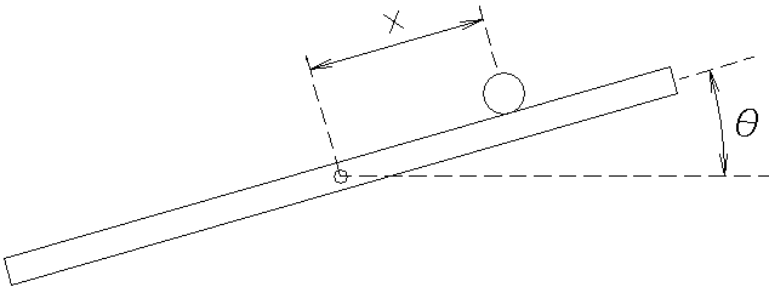
M. Bodson, A. Stolp, 3/22/08
rev,

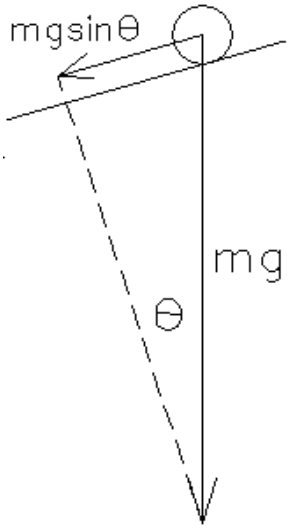
Objectives

The objective of this lab is to give you some experience in the design of a control algorithm. An animated ball and beam system has been programed in Matlab. After trying to control the system manually, you'll design a proportional-derivative controller to do the job automatically.

Introduction

A ball rolling on a beam is both a fun and an educational system.



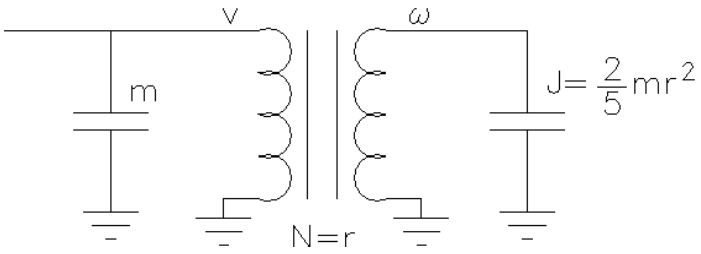


A diagram of a ball on a beam is shown above. The beam is attached to a motor so that its angle θ with respect to the horizontal can be controlled at will and the ball is free to roll under the action of gravity. (You may imagine a small channel in the beam keeps the ball from rolling sideways.) The distance of the ball from the center of the beam is denoted x. The ball can be placed at any location on the beam, and will stay there if its velocity v is zero and the beam angle is zero.

Assuming that the only force acting on the ball is gravity, the force in the x-direction (along the beam) is given by:

$$F = -m\,g\,\sin(\theta)$$

where g is the acceleration of gravity and m is the mass of the ball. This force will accelerate the ball in the -x direction. Because the ball will roll rather than just slide, You'll have to consider both the linear inertia and the rotational inertia (moment of inertia) of the ball. The moment of inertia of a solid sphere is $J = 2/5\ mr^2$. The ball's electrical analogy is shown below.



Show that this can be modeled by a simple sliding mass of $7/5\ m$. Also show the following relationship between the angle θ and the position in the x-direction.

$$\frac{d^2x}{dt^2} = -\frac{5}{7} g \sin(\theta)$$

Unfortunately this is a nonlinear relationship. We will linearize it with the small-angle approximation, that is, replace $\sin(\theta)$ with θ. Now derive the following transfer function:

$$\frac{X(s)}{\theta(s)} = \frac{k}{s^2} \qquad \text{where} \quad k = -\frac{5}{7} g$$

This transfer function is a double integrator, which is not uncommon in control applications when force is the control variable and position is the output variable. It results from Newton's law $F = ma$. Moving a spacecraft with thrusters is another example of such a system. In our case the force is only approximately proportional to θ (the small-angle approximation). Furthermore, in a real system built in the lab our input may have to be the torque of the motor, rather than its position, θ. The model for such a system would be forth-order rather than second-order. Nevertheless, the simplified model will exhibit the most interesting part of the dynamics of the ball and beam system, and is tractable for manual control.

Real-time simulation and visualization

The real-time simulation is programmed in a Matlab macro (m-file). A Matlab figure is used to animate the ball and beam, and you can manually control with a mouse. Download the bbeam_xxx.m file from the class website.

The model of the ball and beam system is implemented in bbeam_xxx.m using an Euler approximation

$$x(i) = x(i-1) + dt \cdot v(i-1)$$

$$v(i) = v(i-1) + dt \cdot \left(-\frac{5}{7} g \sin\left(\theta(i-1)\right) \right)$$

where i is the time instant and dt is the sampling period. (Notice that this model does not use the small-angle approximation– that's only for our control calculations.) The sampling period is set to 0.05sec, or an update frequency of 20Hz. Matlab's tic and toc commands provide the timing. The input and visualization may not occur at exactly 20Hz, but the deviations are not normally noticeable. The animation is drawn in a window using Matlab graphic functions. The window was scaled so that the ball was round on the computer screen used to develop the labs. It may be necessary to readjust the window size on your computer. Only manual control will work when you first run the bbeam_xxx.m file. Automatic control requires other .m files that you will have to write yourself.

the position of the ball is limited to the length of the beam (±0.4m) by setting the velocity to zero when the end of the beam is reached. Friction of the ball rolling on the beam is simulated by setting the velocity to zero if the ball velocity and the beam angle are small enough.

Manual control

Run the bbeam_xxx.m file and select manual control. As an objective, try to move the ball to one of the little red lines and make it stay there. Then try to move it to the other line as fast as fast you can. Have fun and get a sense for the challenges of controlling a double integrator. Notice how you have to account for the ball velocity in your strategy and begin you deceleration early. An automatic controller will need to do the same. Explain the challenges of the manual control problem in your lab notebook and describe the strategies that you have developed for control.

Automatic control

Alright, enough of the kid's stuff, time to automate this baby. Choose a proportional-derivative controller, so that

$$\theta = k_P e + k_V \frac{de}{dt} \qquad \text{with} \quad e = x_{REF} - x$$

As mentioned in the PID lab, there can be problems differentiating x_{REF}, so we'll simplify the control law to:

$$\theta = k_P e - k_V v \qquad \text{where} \quad v = \frac{dx}{dt} = \text{velocity}$$

Derive the closed-loop transfer function. Since there are two feedback paths, you will have to derive it mathematically like you did in the PID lab. This should result in a second-order system whose poles can be placed arbitrarily. Choose the parameters k_P and k_V so that the two closed-loop poles are real and equal, with an associated time constant of 0.3 seconds.

Explain what would happen if the derivative gain k_V was set to 0.

Implement the control algorithm in the real-time simulation. Look for information and comments in the bbeam_xxx.m code about parameters that may need to be modified and calls to other m-files. You should find that you need to create two m-files. An initialization file called bbeamcinit.m which is called once before the simulation starts and a file called bbeamc.m containing the control algorithm. The control algorithm is called at the same rate as the ball and beam simulation. Later you will need to define arrays in the initialization file, and store relevant variables in the control file in order to plot the results of your experiments, but first, just get the system to work.

Your control file should also generate a reference position input that alternates between 0.3m and -0.3m (the positions of the little red lines) every 6 seconds. That is, a square wave with an amplitude of 0.3 and a period of 12 seconds. Something like this could do the trick: $0.3 \cdot \text{sign}\left(\sin\left(\frac{\pi}{6}t\right)\right)$

The variable t in the code is the time. The position of the ball is called xball in the code and the velocity of the ball is called vball in the code. With these you can generate the control signal, theta, the angle of the beam. It should be limited to ±5 degrees. The variable thmax is already defined as 5 degrees in the bbeam file, but you need to do the limiting in your control file.

Once you have your system running automatically, try to run it with the derivative gain k_v was set to 0. Does it behave as you predicted? Try a k_v significantly smaller than your calculated value. How does it behave now? Reset k_v to your calculated value.

Add some lines to your two files so that you can collect data of x, v, and θ for 20 seconds.

Plot the responses of x, v, and θ. You may want to create a third file just for plotting. Include these plots in your notebook.

Repeat the experiment with a time constant of 0.2 seconds and explain the overshoot observed in the responses.

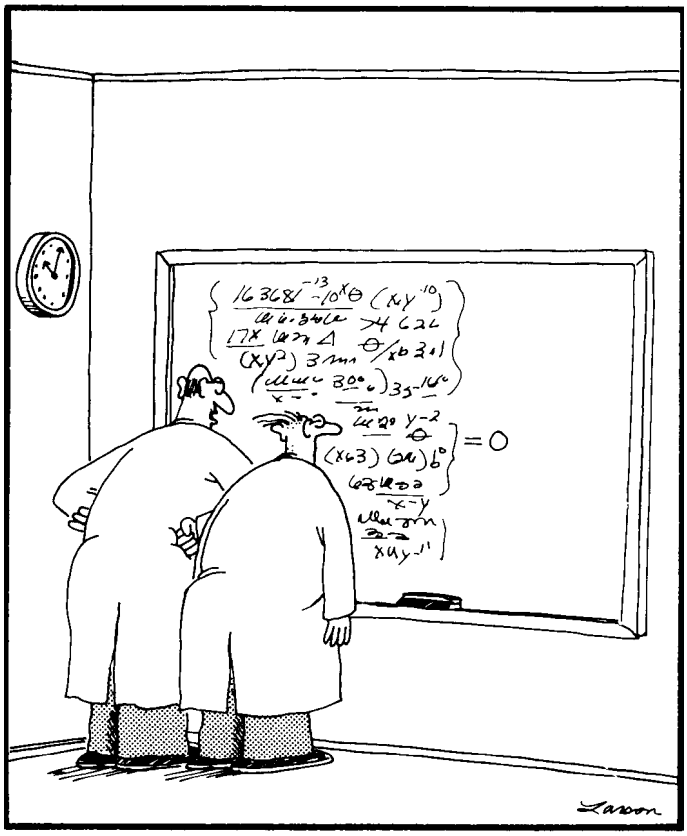
You may wish to try to improve your controller's speed. If you can demonstrate a faster controller (using any control law) to your TA (no fair hitting the end of the beam) your TA can give you extra credit, the faster, the better. Your TA will include a comment in your notebook and will award points later depending on how your speed compares to speeds achieved by your classmates. A position plot is the best way to prove your speed. Speed is measured as the time it takes to get (and stay) within 0.02m of the 0.3m mark. That means that your ball's position must stay within 0.28 to 0.32m.

Insert listings of your bbeamc.m and bbeamcinit.m files in your notebook.

Checkoff and conclusions

Show your code and demonstrate the real-time operation to the TA (either time constant, your choice)***.***

Your conclusion should include some description of challenges and strategies for manual control and a qualitative description of how the automatic controller achieves this. Compare the automatic control to the manual. Include other observations and comments.



"No doubt about it, Ellington—we've mathematically expressed the purpose of the universe. Gad, how I love the thrill of scientific discovery!"